Australian Government
Department of Defence
Defence Science and
Technology Organisation

A Review of Transport Protocols as Candidates For Use in a Tactical Environment

Michael Carter

Information Networks Division
Defence Science and Technology Organisation

DSTO-TR-1808

ABSTRACT

Transport protocols are an essential part of any modern data network, including military tactical networks. Most networks today, including the Internet, operate mainly using the transport protocol TCP. However, this protocol makes a number of assumptions that make it unsuitable for use in a military environment, which is typically characterised by data links that are impoverished in some way. This paper examines TCP, its deficiencies and many other transport protocols that are evaluated for suitability to a military tactical environment. Many of the essential services of TCP as well as other desirable services can be well met by some of these other protocols in existence today in a way that is more suitable for a military tactical environment.

Published by

Information Networks Division
DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh South Australia 5111 Australia

Telephone: (08) 8259 5555
Fax: (08) 8259 6567

AR-013-549
December 2005

APPROVED FOR PUBLIC RELEASE

A Review of Transport Protocols as Candidates for Use in a Tactical Environment

Executive Summary

An essential part of any modern data network is the transport protocol(s) that operates on it. The most well known computer network today is the Internet, which is made up of mostly wired links that are not particularly prone to error. For most of the data transfers over the Internet, Transmission Control Protocol (TCP) is used. As a result, when data networks are extended into new areas, such as over short range wireless links or over satellites, the same protocol, TCP, tends to be used as a kind of default position. TCP in its original form, however, was quite inadequate for large networks, and has subsequently been modified and extended to allow it to operate effectively on the Internet. This has led the protocol to make some fundamental assumptions that make it inappropriate for many different communications environments, and data communications using TCP in these environments perform very poorly. In particular the military tactical communication environment has numerous network links of this type (e.g. satellite and wireless links).
This means that different transport protocols need to be examined to determine if they would be more suitable to be used in military tactical networks. Numerous protocols are examined in this paper to check that they can perform all of the necessary tasks of a transport protocol in an environment with impoverished links.
It is found that there are several protocols that may be appropriate. However, as almost all networked applications use TCP, the candidate protocols are not in wide use. This means that further testing of these protocols using various applications in a military type communications environment would be needed to quantify their performance improvements.

Authors

Michael Carter
Information Networks Division

Michael Carter graduated from the Australian National University in 2002 with a Bachelor of Software Engineering with First Class Honours. He is currently working in the Information Networks Division of DSTO in the area of ad hoc communications

Contents

1. Introduction

This paper is an examination of transport protocols taken in the context of the military tactical communications environment (MTCE).

Transport protocols, which sit at layer four in the OSI model, have traditionally been responsible for managing the traversal of data between user applications and underlying network services. Transport protocols are usually used for end-to-end connections across an inter-network of hosts. The responsibilities of transport protocols have normally been: application multiplexing, reliability of data flow (if desired), flow control and somewhat strangely, network congestion control.

Most data networks in operation today run over stable, wired links that have low error rates and high data capacity. The situation is quite different however, in the MTCE. The MTCE typically has very poor quality links due to a number of reasons, most of them stemming from the need for wireless links. It generally has extremely low capacity (as is common for wireless links). The error rates are generally high, due to local noise, channel fading, shadowing outage and possible enemy jamming. The latencies can be high as the networks are often congested and sometimes require the use of satellite links. The tendency of communications hardware to move causes the link topology to be far from static as well. Finally, the communication channels are rarely duplex (wireless channels by their nature are half-duplex) and sometimes completely one-way (e.g. some stations may operationally required to keep radio silence). These problems are bad enough, but they are further exacerbated by having transport protocols that are not designed for this environment.

By far the most dominant transport protocol in operation today is the Transmission Control Protocol (TCP), first described in [RFC793]. It is the protocol of choice for the Internet today and as such, has been the subject of much discussion in the literature. Consequently, there have been many different improvements, additions and modifications over time. The rest of this report is divided into four parts. The first part is an overview of some of the assumptions that TCP makes and how they might be inappropriate to the MTCE. The second is a rather detailed description of TCP and its de-facto standard variations, including what is used in some popular operating systems. This section also helps to introduce the standard transport layer functions. The third section is a general discussion on some transport layer functions that seem to be inadequate in standard incarnations of TCP. The fourth section is an overview of some of the variations on TCP and other transport protocols that have been proposed to eliminate certain perceived deficiency in TCP performance.

2. Assumptions made by TCP

TCP has been carefully optimised for its intended environment, that is, wired links with low error rates where congestion is almost the sole cause for packet loss. As soon as it is moved into an environment with mobile nodes with error-prone links, some of the design "features" of TCP become undesirable. For example, it does not perform efficiently along network paths which can contain a large amount of data "on the wire" (known as a high *bandwidth-delay* product)[RFC1323], especially if the receiver window size is small by comparison (see [KHUU04]). Throughput is also seriously degraded by random errors when the product of the loss probability and the bandwidth-delay product approaches or becomes greater than one, causing problems for high-speed (e.g. gigabit networks), high latency (e.g. satellite links) or highly error-prone links (e.g. wireless links)(see [LAKS97]).

While TCP connections attempts to be fair with co-existing TCP connections, TCP will in fact, favour those connections with small round trip times.

The most popular TCPs also assume segment losses are the result of congestion. If, however, losses occur due to link errors, or come as a result of node mobility, then congestion control mechanisms will be incorrectly invoked, leading to further degradation of performance (see [LEFE00]). This is a particular problem for connections that pass over wireless links. Indeed, in mobile networks (with frequent packet errors and route failures due to mobility), TCP will tend to operate more in the slow-start phase than the congestion avoidance phase. This results in both unfair traffic and under usage of network resources.

Most studies of TCP performance over wireless links generally only consider the wireless link to be the last link in an otherwise wired network path (i.e. like a mobile phone connection). In fact TCP performance over *multiple* wireless hops is much worse than on paths with a single wireless link, due to (and of course, depending on) interactions with wireless MAC protocols. Indeed, in the absence of link layer error control, it becomes impractical to use window sizes larger than one segment (see [GERL99]).

Most TCPs also assume a reasonably constant round trip time for traffic, or at worst, a round trip time that depends on network load (carefully governed by congestion control mechanisms), which may not be the case, particularly in mobile networks with rapidly changing topology. A similar problem arises from the traffic in one direction relying on characteristics of the traffic path in the reverse direction (on which acknowledgements come through). In ad hoc and mobile networks, the forward and reverse paths may not be the same.

3. TCP – A Description

3.1 Basic TCP

TCP is a transport protocol designed to run over the top of an unreliable packet based network, typically over an IP network. Its goal is to provide reliable, ordered streams of data between two hosts. These two points will be referred to from now on as the *sender* and *receiver*. Although both points can send and receive data, it is easier to consider the data flow in one direction and keep in mind that the two hosts have symmetric capabilities. Therefore, we would like a sender of data to be able to send some arbitrary amount of data which can then be read by a receiver with no errors or omissions, and with as small a delay as possible. Despite appearing to users as a byte stream uninterrupted by artificial boundaries, data is passed by TCP in discrete segments, each of which has a TCP header pre-pended.

The original TCP specification concerned itself with providing three main things: multiple connections, reliable data transfer and receiver-side flow control. How each of these is provided will be briefly described.

3.1.1 Multiple Connections

All TCP data transfers take place within the context of a *connection*, which is set up between two hosts, and can exist for an arbitrarily long period of time. Before any true data transfer can occur, the connection has to be set up by way of a *three-way-handshake*. This is just a short exchange of packets that allows both ends of a connection to confirm that there is another host wanting to set up a connection, and that they themselves are willing to set up a connection. It is also the time when any TCP connection options are negotiated. The original specification only defined one real option, and that was to change the *maximum segment size* used by TCP. It is desirable for more than one TCP connection to be able to exist on a single host at once, to allow for a single host to be "connected" to multiple other hosts and/or to have multiple streams of data between two hosts. To this end, each TCP connection is identified by both the network addresses of the two hosts and an additional identifier on each host called the *port*. As the source and destination port numbers are in the header of each TCP segment, each segment can be correlated with the appropriate connection. (Note that the connection is abstract, it only exists in the records that are kept about it at either host.) If a segment arrives at a host that does not appear to be associated with a valid connection, TCP will respond by attempting to inform the sender of the invalid segment that it believes an invalid connection to be valid.

As each side of the connection finds it has no more data to send, it can close the connection, which consists of sending the other side a notification that it has no more data to send. A TCP connection is not fully closed (and hence cannot be expunged from the memory of the host) until both sides have indicated they wish to close it.

3.1.2 Reliable Data Transfer

Reliable data transfer has two parts. The first is to make sure that data arrives in the same order that it was sent. The second is to make sure that there is no missing or extra data.

To ensure data is kept in the correct order in which it was sent, every byte sent by TCP has its own individual 32-bit *sequence number*. Receivers cannot pass data to users of TCP if it is missing bytes of data from its stream. Each TCP segment sent has in its header the sequence number of the first data byte in its segment. This is enough to indicate the sequence numbers of every byte in the segment. The initial sequence number is not zero or one but rather is selected by the sender at the time the connection is set up. This mechanism is enough for TCP to know the correct order in which data was sent. It does however, have some subtle problems when operated over a packet network that can duplicate and delay packets. A discussion of these problems is outside the scope of this document, for more detail see [TANE03].

Ensuring that *all* the data sent gets to the receiver is achieved through the use of an acknowledgement number that is in the TCP header of all segments going back from the receiver to the sender. This number corresponds to the sequence numbers in the TCP header sent from the data sender. A receiver that sends a valid TCP segment with a given acknowledgement number is indicating it has correctly received all data up to but not including the acknowledgement number. A sender will expect acknowledgements for the segments that it sends. If it does not get them, because either the data or the acknowledgement for the data was lost on the unreliable network, then the sender will have to re-send the data. To achieve this, each segment transmitted has associated with it a *re-transmission time-out* that is used to indicate when a sender should re-transmit a data segment. The timeout, to be as efficient as possible should be related to the expected time it would take for the segment to reach its destination, and for an acknowledgment to get back to the sender (i.e. the *round trip time*). As different network paths will have different latencies and network conditions may change over time, an estimate for the round trip time (and also the re-transmission time-out) has to be continually updated. The original TCP specification suggested updating the estimated round trip time (RTT) every time an acknowledgement arrives according to the formula

$$\textit{new estimated RTT} = (a * \textit{old estimated RTT}) + ((1-a) * \textit{segment RTT})$$

where $0 < a < 1$ is some value to control how quickly the estimated RTT will change. The re-transmission time out is then suggested to be

$$\textit{re-transmission timeout} = b * \textit{estimated RTT}$$

where $1 < b < 2$ is an arbitrary constant. [TANE03]

3.1.3 Receiver-Side Flow Control

As convenient as it would be to have a receiver that can handle an infinite amount of data, it is never going to happen. It is clearly wasteful for a sender to inject more data into the network if the receiver will not be able to store it. To control the flow of data from the sender, TCP uses an advertised *sliding window*. In any segments sent back from the receiver to the sender, the header will contain a *window size* that indicates the amount of data that the receiver is willing to accept. As the sender knows how much data it has sent, and how much data has been successfully acknowledged, it can simply refrain from allowing more unacknowledged data onto the network than can be handled by the receiver. A receiver can indicate a window size of zero, effectively restricting the sender from sending any more data until the receiver indicates otherwise. This could create deadlock, however, if a receiver increases its windows size from zero, and the indication gets lost on the network (segments that have no data, only acknowledgements, are never re-transmitted). For this reason the receiver is always prepared to accept a one-byte data segment, and the sender can send a one-byte segment if it thinks the receiver's window is zero and has not received an update from the receiver in some time.

3.2 Problems with the original TCP and the standard variations

As TCP was put into use on real networks, and people gained more experience with its behaviour, a number of problems were discovered. The following is a description of the proposed solutions to these problems as well as other assorted suggested improvements to increase TCP's efficiency. These improvements were suggested over a long period of time, and not all implementations used all improvements, and this led to a wide variety of TCP "versions". The following is roughly organised along the lines of some of the better known "versions".

One of the first problems discovered with TCP was the so-called *silly window syndrome*. In this situation, a receiver would have indicated a window size of zero, meaning that it could not accept more data until it had had the chance to process some of the data it already had. It would then proceed to process 1 byte of data, and send an acknowledgement back to the sender with a new window size of one. The sender would then send a segment with one byte of data to fill up the window. The cycle would then repeat. The solution to this problem, proposed in [RFC813] is simply to prevent the receiver from sending a new window update until it can handle a full maximum segment size or half its buffer is able to be filled. Likewise the sender shouldn't send until it can send a maximum segment size or fill half the receivers buffer. This fix is standard to all implementations of TCP.

3.2.1 TCP Tahoe

A large number of improvements were suggested in [JACO88], and these were first included into what is commonly called *Tahoe TCP*. All of these improvements are based on the idea of normal traffic being an almost "steady-state" transmission where data is

flowing from the sender to the receiver at a constant rate, and segments are placed into the network at the same rate that they are being removed from it.

One improvement suggested was an improvement to the setting of the re-transmission timeout measure. The improved method involved using some quick methods to estimate the variance of segments' round trip times, and base the timeout on a combination of the estimated round trip time and the measured variance. Despite this improvement, it is still necessary to double the re-transmission timeout for a packet each time it is re-transmitted. Otherwise an increase in network delay would cause some packets to always be re-transmitted, possibly multiple times.

Ideally, we would like the sender to be able to send off a full receiver windows worth of data at once, and then send more data as is found to be appropriate from returning acknowledgements. This is in fact what earlier TCPs attempted to do. However, just because the receiver can handle a certain amount of data doesn't mean the intervening network can. It may not have the capacity, or other users and/or hosts might be using it. In fact, it was found that TCPs that behaved in this way not only had to be constantly re-transmitting segments that were inevitably dropped, but it also seriously degraded the performance of other unrelated TCP connections operating over the same network. The sudden excess of data on the network could immediately overload the network and cause packet drops, or it might cause the average round trip time of other TCP connections to drastically increase because much more time from intervening routers would have to be dedicated to processing the excess. Which of course caused other TCPs to re-transmit segments unnecessarily and only made the problem worse. The solution is to increase the amount of data a TCP connection would put onto the network only in response to acknowledgements returned from the receiver. That way, if acknowledgements are slower to come back, then the TCP will be slower to increase its transmission rate, and give time for the estimates of round trip time to increase with network load. The policy – called *slow-start* – is to create a send window in addition to the receiver window, with the sender never transmitting more than the minimum of the two windows. The send window is initially one or two maximum sized segments (MSS) and is increased by one MSS for each segment that is successfully acknowledged.

Of course, this is an exponential increase in the sender's window size, which will surely eventually cause congestion if it continued forever. For this reason, an additional *congestion avoidance* strategy was suggested. This strategy requires a *slow start threshold,* which is an upper bound on the sender's window for which the slow start is used. Once beyond this threshold, the senders window is only increases by one MSS each time a complete windows worth of data has been acknowledged. With a more accurate round trip time estimate, it is then reasonable to assume that a re-transmission timeout occurring indicates a packet has been lost. If it is assumes that this is caused by congestion, then TCP needs to lower the amount of data it is putting on the network. It does this by cutting the slow start threshold in half and reducing the sender's window to one or two MSSs, which will induce a return to the slow start algorithm. For a more detailed justification of these algorithms, see [JACO88].

Another improvement, first suggested in [KARN87] is to deal with a subtle problem with estimating the round trip time. If a segment is ever re-transmitted, then when a sender receives an acknowledgement of that segment, they cannot tell if that acknowledgement was for the re-transmitted segment, or the original segment that was delayed for some reason beyond the normal round trip time. Of course, TCP could just not include re-transmitted segments in its round trip time estimation, but consider what would happen if there were a sudden increase in round trip time, perhaps due to a router crashing. Then most segments might have to be re-transmitted, but there will not be any new estimates for the round trip time, causing the re-transmission timer to always be too short. The solution is to not only double the re-transmission timeout for the re-transmitted segment, but also for any subsequent packets, until something gets through without being re-transmitted.

The last improvement, commonly called *fast retransmit* is based on the observation that there may be a way to tell if a packet has been lost before the re-transmission timeout. The idea is that if you send off a sequence of segments, each with increasing sequence numbers, then you should get, in return, a sequence of acknowledgements, each with increasing acknowledgement numbers. TCP can only acknowledge a byte if it has received every byte up to that one. So if a sender receives a sequence of acknowledgements (generated in response to the receiver receiving a sequence of segments) which all have the same acknowledgment number, then it is reasonable to assume that a segment (indeed, the segment starting with the sequence number corresponding to the repeated acknowledgement number) has been lost. Once this has been worked out, the sender can immediately re-transmit the segment and reset the re-transmission timeout. The suggested improvement to TCP implementations was to wait for three consecutive duplicate acknowledgements and conclude from that that congestion has occurred and a re-transmission is necessary.

These improvements make TCP Tahoe a significant improvement on the original TCP specification, especially when there are many TCP connections on a given network. It is worth noting, however, that the congestion control mechanisms outlined above are essentially co-operative, which means there is nothing to stop an inconsiderate or perhaps just poorly implemented transport protocol from flooding the network with too many packets and causing co-existing TCP connections to perform poorly.

3.2.2 TCP Reno

TCP Reno introduced a further optimisation on top of the fast re-transmit algorithm. Jacobson stated that when congestion control is performed as a result of receiving duplicate acknowledgements, it is unnecessary to perform a full slow start (see [JACO90]). Instead, he suggests a *fast recovery* whereby instead of reducing the sender's window to one or two MSSs, it is instead reduced to the new slow start threshold, plus one for each of the duplicate acknowledgements received. The reason for adding one for each acknowledgement received is the assumption that segments arriving at the destination,

despite those segments not having been acknowledged, create the duplicate acknowledgements. TCP can therefore infer that the sender's window should be that little bit extra bigger. This modification to the window size must be removed when any new acknowledgements come through, as a new acknowledgement would indicate information newer than any of the duplicate acknowledgements.

3.2.3 Other Variations

There are some other variations that are sometimes included in TCP implementations. One is the keep-alive timer, which is simply a timer that, when it goes off, will cause TCP to send a small data-less packet to the other side of the connection, just to make sure that it is still there and has not crashed. Another idea is to delay acknowledgements for some small fixed period of time after segments are received or some small number of segments in order to save on the total number of acknowledgements that are sent back over the network.

3.2.4 TCP Options

Various options have been proposed over time to extend the behaviour of TCP. Usually these have been to improve the efficiency of TCP over very high speed links and/or links which typically have large amounts of data in transit at one particular instant. These options include:

- Window scaling – which allows all sequence and acknowledgement numbers and all window sizes to be interpreted as multiples of bytes, rather than individual bytes.

- Time stamping segments – which involves a sender placing a timestamp on each TCP segment, and the receiver echoing that timestamp on the associated acknowledgement. This allows for improved measurements of round trip times, and hence better protection against sequence number wrap-around.

- Selective Acknowledgements – which allows a receiver to specify blocks of data that it has received that otherwise wouldn't be represented in the acknowledgement number. This allows for more efficient re-transmission by the sender.

3.3 TCP in some common operating systems

Most TCP implementations today are based on TCP Reno, which includes slow-start, congestion avoidance, fast retransmit and fast recovery, as well as Clark's, Nagle's and Karn's algorithms.

3.3.1 Windows

The Windows 2000 TCP implementation contains all the TCP Reno features that you would expect. It also contains support for the window scaling option, the SACK option and TCP timestamps. It also implements delayed acknowledgements, path MTU discovery and uses keep-alive messages.
Windows 2003 and XP include all of the above as well as automatic negotiation of window scaling and TCP timestamp options and a newer IGMP version.

3.3.2 Linux

Linux TCP implements all of the features of TCP Reno, as well as including some other features. Linux allows the use of both SACK and the NewReno modifications. It supports window scaling, TCP timestamps and the use of the D-SACK and FACK options. Delayed acknowledgements are also used. Linux TCP includes some unusual elements as well, such as the use of a more fine grained timer than is required by the IETF standards. Linux TCP is also able to use Explicit Congestion Notification and has a mechanism to undo congestion window adjustments that are made if it later detects they were unnecessary (e.g. congestion was inferred from duplicate acknowledgements but the segment was only delayed, not lost, and arrived soon after).

4. Desirable Transport Protocol Features

TCP has some good elements, but is inherently lacking some of the elements that would be desirable in a transport protocol for the MTCE. The following is a discussion of some of the important elements of a transport protocol that are inadequately or inappropriately addressed in TCP.

There are two transport protocol concepts that have become confused, and the responsibility for confusing them can be laid directly at the feet of TCP. These two ideas are flow control and rate control. TCP began its life with flow control, the idea that a data sender should not send more data than the receiver can buffer (implemented through sliding windows), as this would cause data to be lost unnecessarily. Far more important is the concept of rate control, which in its coarsest form is simply limiting a data sender to send only a certain amount of data in a certain timeframe. Rate control operates on two levels. Firstly, on a network level, that is, a data sender should not send more data than

that which would be able to be handled by the intervening network infrastructure. Secondly, on an application level, that is, users may wish to regulate the data rate that an individual application might use. TCP includes a limited version of the first form of rate control and doesn't address the second at all. The congestion window in TCP and the associated control algorithms are a confused attempt at rate control that attempts to use a flow control like mechanism to address a fundamentally different problem. This has led to all sorts of problems (e.g. the 'ack-clocking' to stream data into the network is inappropriate for high speed or long delay networks) and has been the source of many attempts to improve TCP by adjusting the way in which the congestion window is altered during operation (see section 3.2). Unfortunately, network layer rate control is a network layer problem, and to be properly solved needs to be addressed at, or in conjunction with, the network layer. This can be clearly seen from TCP's congestion control mechanisms (which are an attempt at rate control) getting confused by the difference between congestion and link errors. Both of these are problems that exist at the network layer or below and if transport layers wish to take account of these, then they require some support from lower layers to distinguish the difference.

This is even more complex if you also wish to address the problem of *fairness* (also called relative rate control). It is insufficient to rely on protocols having been designed to be fair between data flows. To start with, it is not possible to ensure that polite protocols will be used (UDP, a common transport protocol, has no concept of rate control and TCP itself is only fair to other flows if they have the same round trip time). But even if this were possible, there would still be no mechanism to distinguish between flows, so misbehaving applications could still act unfairly by using several different flows (e.g. some peer-to-peer applications use multiple TCP connections to seize extra data capacity) Again, this is an issue that can and should be controlled at the network level. However, TCP has shown that it is not necessary to rely on the network layer to solve network layer problems. Indeed, many transport protocols have a better form of rate control than TCP without relying on extra information from the network layer. Examples of these include XTP (see section 5.7.2), WTCP (see section 5.9.6) and ATP (see section 5.5.2). Some other protocols, sech as XCP (see section 5.10.1) include the idea of network support within them, leading to more accurate rate control. Particularly in capacity constrained environments like the MTCE, rate control (in some form or other) is an important element.

Reliability is another important aspect of a transport protocol. The TCP mechanism of using a re-transmission timeout for unacknowledged segments has some subtle problems. It heavily relies on accurate estimations of round trip times, which may not be possible in environments with highly varying latencies. For example, it doesn't work well with reliable data link schemes in wireless networks as their time to deliver segments may change drastically depending on varying local conditions. A better method that was in part introduced to TCP, and is indeed used by most protocols that include reliability, is to rely instead on selective and negative acknowledgements from the receiver. This is seen in protocols like XTP (see section 5.7.2), FITFEEL (see section 5.3.2), P_MUL (see section 5.3.3) and WTCP (see section 5.9.6). Loss of acknowledgements can in general be addressed through periodic sending of acknowledgements from the receiver.

5. Different Transport Protocols

This section contains brief descriptions of some of the transport protocols that have been proposed over time and some thoughts on their suitability for the MTCE. It also contains some information on ideas that aren't transport protocols themselves but could be useful in solving the problems seen with existing transport protocols. This section is arranged roughly along the lines of the specific problem the individual protocols were attempting to solve.

5.1 Explicit Congestion Notification

Many of the variations on TCP and other protocols are based around improving the accuracy of TCP's congestion control mechanism. As network congestion is essentially a network layer problem, TCP relies on guessing the network condition based on how it perceives segment flow. Explicit Congestion Notification (ECN, [RAMA00]) is a scheme to provide information from the network layer on its state of congestion to the transport layer. This scheme must be present on all of the intervening routers at the network layer. Basically, it begins to mark network packets (eg. with a bit in the IP header) with a congestion warning if the router's buffers are becoming close to full. The receiver of the marked packet can then echo the congestion warning back to the sender by marking a bit in the TCP header, then the sender can take appropriate action.

5.2 TCP Optimisations

The following section describes some of the more exotic varieties of TCP that essentially try to improve its performance without addressing any of the fundamental problems that it faces. Unfortunately, this means that they really won't be suitable for the MTCE.

5.2.1 TCP NewReno

A slight improvement to TCP Reno suggested in [RFC2582], gives us a slightly different implementation of the fast recovery algorithm. This suggestion, called TCP NewReno, observes that the standard fast recovery technique really relies on single packets being lost. As there are often occasions when multiple consecutive packets or packets in close proximity are lost, a change was needed. The new algorithm requires TCP to record the highest sequence number sent when a segment drop is detected with duplicate acknowledgements. That way, when new acknowledgements finally come through, TCP will be able to tell if all the data sent before the time the lost segment was detected actually got to the receiver or not. If the new acknowledgement doesn't indicate this, then it is assumed that another segment was lost (which packet can be determined from the new acknowledgement number) and the senders window is adjusted in-line with the original fast recovery policy, and not reduced to the slow start threshold, as would have happened in TCP Reno. Note that this solution is in some ways similar to the selective

acknowledgement (SACK) option for TCP. Indeed, if SACK is used, then the NewReno improvements are not needed.

5.2.2 TCP Vegas

TCP Vegas, described in [BRAK95], is essentially a modification of Reno. It simply optimises three of the Reno features. The first is a method to re-transmit lost segments even faster than the fast re-transmit algorithm. Basically it uses timestamps on TCP segments to detect lost segments more quickly than would otherwise be found by waiting for n duplicate acknowledgements. The second is an improved congestion avoidance technique. It adjusts the sender's window up or down depending on how the actual traffic throughput is varying from what the sender expects. To do this it has to maintain an estimate of the currently available bandwidth to each data flow. Unlike earlier TCPs, this doesn't rely on creating congestion before actually doing something about it. The final tuning is an adjustment to the slow-start algorithm whereby the growth rate of traffic is halved. The reason for this is to allow TCP Vegas to test the data rate it is receiving against what it would expect. If it finds that the actual data rate is dropping, then it will switch to a linear traffic growth (i.e. congestion avoidance). This is intended to prevent segment losses due to congestion before they happen.

5.2.3 TCP Veno

TCP Veno is named after Vegas and Reno, the two TCPs that it is described to be a combination of [FULI03]. The main idea behind this TCP is to address the problem of being unsure whether a segment lost is due to congestion or link error. Veno's only modification to Reno is to take a bandwidth estimator similar to the one described by Vegas, and use this to make guesses about the cause of segment losses. Specifically, if a segment is lost when the estimated available bandwidth is not being fully used, the lost segment will be considered due to link error and the sender's window will not be reduced by 50%. To be conservative, the sender's window will still be reduced, but by a smaller margin.

5.3 UDP based

The protocols in this section attempt to achieve some desired elements of transport protocols without dealing with the existing complexities of TCP. They do this by mainly using UDP (a *very* simple transport protocol) and extending its use in some way. Note that for some protocols (e.g. P_Mul), using UDP is a convenience rather than necessity – the protocol may be implemented on IP directly if it is widely accepted as a standard in many operating systems.

5.3.1 RBUDP

Originally intended for high-speed bulk data transfers, the reliable burst user datagram protocol (see [HELE02]) is a hybrid of UDP and TCP. RBUDP attempts to sidestep the

problem of TCP windows sizes restricting the data sending rate. It does this by sending all its data via UDP at a user-specified rate rather than using TCP. In this way, it is not restricted by TCP mechanisms. UDP is unreliable, so RBUDP uses a simultaneous TCP connection to retrofit reliability. When a sender has sent all its data, it indicates this with a message on the TCP connection. The receiver will then respond with an indication of the UDP packets that it has received. If any are missing then they will be re-sent via UDP, and the TCP connection will again be used to check delivery of the data. Note that this scheme has no congestion control, indeed it is intended to have no congestion control to get in the way of the data rate that a user may wish to send. It also avoids having to deal with acknowledgements coming back the other way for most of its data transfer phase.

This characteristic in particular is an idea that might be adapted for use in a wireless network, where it would be useful to avoid acknowledgements as they add to the amount of contention in the network. Contention being a much more serious issue in wireless networks than wired. The problems with TCP also generally wouldn't appear if the TCP traffic were small enough to be restricted to one segment at a time. Although this would naturally incur a possibly large time penalty in transmitting data. RBUDP would be sufficient for non-time critical data.

5.3.2 FITFEEL

FITFEEL (See [KENN97]) is a UDP based protocol designed specifically for highly errored links. Data to be sent is divided into messages that are placed into UDP payloads (there may be more than one message per payload). These are sent at a fixed rate of transmission without any guarantee of delivery. There is a facility for the receiver of data to send acknowledgements and negative-acknowledgements back to the sender to help the sender decide which data it should be sending. Essentially though, the sender just keeps transmitting data at a constant rate until the receiver tells it to stop. Likewise the receiver keeps telling the sender to stop until it actually does stop. Implementation details such as when and how often to send acknowledgements is not specified.

This protocol has the advantage of being exceedingly simple. It doesn't require excessive control information to be passed between hosts so it could well be used for asymmetrical links. One drawback is that it uses a fixed rate for sending data. For a single link, the best rate could be learned or tuned from experience. For a complex internetwork, having no rate control could be a serious drawback. This suggests that FITFEEL would be best placed as a single link only protocol, or possibly integrated into a more complex transport protocol that uses proxies.

5.3.3 P_MUL

P_MUL is a protocol that combines many features. It stitches together multicast delivery, reliable UDP based data transfer and the rather unusual feature of specifically tailoring itself to situations where the receiver cannot acknowledge data (for example, if the receiver must maintain radio silence) for an extended period of time. The multicast groups

are globally handled through a unique multicast address that must be known to all P_MUL users. This address is used to provide information on which nodes wish to transmit to which destinations and hence form other (temporary) multicast groups. It is also through this group that information on wether or not a particular node can respond (for example, whether or not the node is attempting to maintain radio silence) is distributed. Note that dynamic formation of these groups generally requires two-way traffic, and hence should be formed before nodes are likely to become silent, or statically assigned (dynamic formation of groups is omitted in ACP 142). Data is sent in units called messages consisting of a number of data packets based on UDP. Individual P_MUL segments contain sequence numbers so the receiver can identify segments that are missing from messages.

For receivers that can respond, each receiver will either acknowledge a message has been completely received, or indicate back to the sender the missing segments, which will be re-transmitted. If no acknowledgements are forthcoming, the sender will re-transmit segments after a timeout period.

For receivers that cannot respond, it is expected that they will eventually be able to respond and signal to the sender what messages they have received, complete or not. In the meantime, the sender maintains another timer specifically to cater for the silent receivers. Whenever this timer expires, it is reset and the complete message is sent again to all of the silent receivers. The number of times this occurs is set by a parameter. (See [ACP142] for a complete description of P_MUL)

The support for inactive receivers is an interesting feature of P_MUL, but could probably be adapted into any other transport protocol along with the multicast facilities. The reliability features are simply a UDP protocol with selective acknowledgements. This is useful, but not astounding.

Forward error control (FEC) coding can be added to P_Mul, as to other multicast protocols, to improve the throughput efficiency [TOZH04].

5.4 Other Message Based

Many protocols that desire a message-based service rather than a stream-based service base themselves off UDP. However this is not always the case and some message-based protocols take a different path altogether.

5.4.1 SCTP

The Stream Control Transmission Protocol, described in RFC 2960, is a transport protocol designed to reliably deliver data (i.e. without loss or duplication) while maintaining as much order in the data as is required. It is, however, not a TCP variant. But it does show many similarities to TCP with SACK, such as being a connection oriented protocol that uses the same algorithms and sliding windows for congestion control and flow control as

the more modern TCPs and it also uses similar methods to govern the use of SACK segments. It is probably best described as how it is different to TCP though.

SCTP is message oriented. That means that individual segments of data are delivered to the STCP user in exactly the same form that they are sent. In much the same way as UDP there is no notion of a continuous flow of bytes, just messages. SCTP messages though, do have their order preserved at the granularity of a stream. Each SCTP connection can be divided into multiple streams, each which logically represent (possibly) unrelated data. This means that two segments in different streams do not have to have their relative order preserved and that one segment does not have to be present for the other to be delivered to the STCP user. This can produce some benefits in reducing delays.

Another interesting feature of SCTP is the multi-homing ability designed into the protocol. Most transport protocols satisfy themselves with delivering data from one network address to another, with the implicit assumption that a single host will have a single network address. SCTP allows that a host can have multiple network addresses and hence leaves open the possibility of having multiple *different* network paths from a data sender to a data receiver. SCTP does this to provide redundancy in case of network failure. Multiple source and destination network addresses are established at the time a connection is initialised. SCTP maintains a check on each path it establishes by sending periodic heartbeat segments to every address it thinks it can send to.

Security was a significant concern in designing SCTP and so some care has been taken to protect SCTP from those well known types of attacks that can degrade TCP. See [RFC2960] for further information on this aspect of SCTP.

SCTP has some unique features. It was designed for the use of telephony over IP networks and so its best application would be to problems that require similar sorts of traffic. The multi-homing capability sounds like a good idea in principle, but to gain any real use from it, there would actually need to be different network paths from one host to another. Particularly in a mobile environment, the slight redundancy is unlikely to justify the difficulty of finding multiple paths between hosts. The multi-streaming idea to reduce overhead is an excellent idea, but it is only really applicable if data going to a host can be usefully separated into logically different streams. Whether or not this is a useful feature would then depend heavily on the application of your data transfer.

5.5 Ad hoc Networks

The protocols in this section have been explicitly designed for use in ad hoc networks. As the MTCE typically has many characteristics similar to ad hoc networks, transport protocols designed for ad hoc networks probably have the greatest potential for use in the MTCE.

5.5.1 ADTCP

This variation of TCP is built upon TCP NewReno, but is designed for use in ad hoc networks. It begins with the premise that packet losses may be cause by other factors than congestion, particularly in ad hoc networks. What ADTCP adds is simply a more sophisticated way of determining the cause of packet loss, it doesn't actually change the TCP response to congestion. It does not, however, require support from intermediate routers to do this. ADTCP achieves this by keeping measurements of four values that are not kept by a standard TCP. These are: the Inter-Delay Difference, the Short Term Throughput, the Packet Out-of-Order ratio and the Packet Loss Ratio. When congestion would otherwise be inferred (when the re-transmission timer goes off or 3 duplicate acknowledgements are received), ADTCP examines these four parameters to make a decision on what caused the problem. It will conclude one of the following: congestion has occurred, there has been a link error, or the route to the receiver has changed. Depending on which event ADTCP concludes has occurred, it will take appropriate action. (See [ZHEN02] for more details on the connection between the four variable states and the "unusual" event)

This could turn out to be a very useful upgrade to TCP if it can accurately determine the reasons for packet loss as it does not require explicit support from the network layer. It is still essentially a work around though, and doesn't address TCPs other problems. Of course, addressing those problems may turn out to be unnecessary.

5.5.2 ATP

The Ad hoc transport protocol, described in [KART03], arose out of examining the deficiencies of TCP in an ad hoc environment. Rather than try to adapt TCP, ATP attempts to be a transport protocol that doesn't contain the deficiencies to begin with. One of the essential features of ATP is to use a rate-based transmission mechanism, rather than a window-based mechanism. This helps to avoid bursty traffic, which is undesirable on an ad hoc network. It also removes the reliance of acknowledgements to control traffic flow, and hence helps to avoid under use of the network resources. In fact ATP doesn't use ordinary acknowledgements at all, it relies only on selective acknowledgements from the receiver to ensure reliable delivery of data. Congestion control is handled by an entirely different mechanism. ATP requires that each intermediate node in an ATP route to maintain the average queuing delay and the average transmission delay experienced by segments flowing through that node. Each segment is stamped with the maximum of the sum of these delays as it passes through the network. An ATP receiver can then use an average of these values to provide feedback to the ATP sender as to what rate it should be transmitting at. Note that ATP can avoid the specific problem addressed by the sliding window in TCP (i.e. that the receiver application may not be able to process data as quick as the sender can send it) by observing the rate at which the receiver application is accepting data and accounting for that value in the feedback provided to the ATP sender. An ATP sender will calculate a new rate from the feedback and adjust its sending rate accordingly, with two caveats. Firstly, it will only increase its sending rate by a fraction of

the available increase, because extra segments on the network can reduce network capacity by more than their own size due to increased contention. Also, there is a small threshold that an increase must be larger than to be allowed as an actual increase. This is simply to prevent insignificant changes to the sending rate.

This protocol has made the right move in not totally basing itself on TCP and all the inherent historical left overs in that protocol. This will be effective as it is addressing congestion concerns where they occur, at the intervening routers. Obviously, this requires more interaction between network and transport layers. That may complicate the interface between them, but could be offset by simplifying the transport protocol. Especially since in an ad hoc network (and possibly the MTCE) you would expect every participant to have similar communication stacks. You would not then see intervening routers being able to get away with only having a network layer implementation.

5.5.3 TPA

While purporting to be something different, the Transport Protocol for Ad hoc networks (see [ANAS03]) is still essentially a TCP derivative. The modifications and additions it makes are however, specifically designed for ad hoc networks. The first change it makes is an attempt to address unnecessary retransmission. Segments are managed in blocks of a defined size. All segments in these blocks need to be received before another sending another block of segments will be attempted. The segments are acknowledged individually by way of a bitmask included in each segment header. This acts as a kind of selective acknowledgement that allows re-transmissions to be saved up, which in turn allows more time for acknowledgements to arrive if they have been delayed by a changing network topology. Another change is designed to handle route changes. When a route to an end host is lost (possibly indicated explicitly from the network layer), data transmission is frozen until a new route is established, and the round trip time and variance estimators are adjusted to heavily favour newer samples of round trip time in order to quickly remove data relating to a link that no longer exists. Finally, TPA introduces a small threshold of consecutive re-transmission timeouts that must be passed for TPA to infer congestion and perform congestion control.

This appears to be more of an optimisation of TCP than a new protocol. While it is probably effective in what it sets out to achieve, it does not address the fundamental problems of TCP.

5.5.4 ATCP

Ad Hoc TCP is an add-on to TCP that essentially tries to remove some of the flaws of TCP on ad hoc networks, without requiring extensive changes to and maintaining compatibility with existing TCP implementations. It is implemented as a layer between existing IP and TCP layers. ATCP intercepts segments that are incoming to TCP and makes decisions on what the TCP implementation should receive in order for it to behave sensibly. When ATCP detects what would be interpreted by TCP as congestion (3 duplicate

acknowledgements or a re-transmission timeout), ATCP will force TCP into persist mode and re-transmit lost segments itself. Similarly, when a route to the destination is lost, ATCP will force TCP into persist mode. These measures are to ensure that TCP does not invoke congestion control when it is not necessary and that it does not terminate a connection when a link may only be temporarily unavailable due to node mobility. The only time ATCP will allow TCP to use its congestion control mechanisms is when it is explicitly notified on network congestion by something like ECN. For more detail see [LIUS01].

ATCP is a neat work-around that appears to try and anticipate sub-optimal behaviour from TCP and trick it into performing correctly. It would be interesting to see how well patched up versions of TCP like this compare to fundamentally different transport protocols in an MTCE like environment.

5.6 Transactional Methods

Transactional protocols are intended for short, possibly one off messages between hosts. Their design typically attempts to minimise the time taken for an individual transaction between hosts.

5.6.1 T/TCP

One cause for concern with standard TCP implementations is the fact that before any actual data is sent over a newly established connection, a three way handshake must complete (This also helps to prevent duplicating transactions due to network duplication of packets). This will add at least an extra round trip time to any TCP connection. Although this may not seem like much, it is very inefficient if the TCP connection is only being opened to perform a short request/response like transaction. Transactional TCP, described in [RFC1644], is an extension to TCP that allows complete transactions in only three sent segments. To do this, it works around the standard TCP three way handshake. Typically, a TCP segment is sent that has both the SYN and FIN bit set, with all the data in the request in the first segment. However, doing this also works around the protection against the network duplicating packets. Therefore T/TCP also requires the use of a new TCP option (the Connection Count) that basically just acts as an additional identifier to the senders network address and port. The receiver can then detect duplicates by storing the connection count for each distinct sender. This mechanism can also help to reduce the time required for the connection to remain in the TIME_WAIT state after finishing data transfer. This in turn allows the communication endpoints to be quickly re-used by other T/TCP requests.

An elegant way of avoiding the time penalty in the three way handshake like this one would probably only be useful if two conditions hold. Firstly the data sent has to be very small (enough to fit into one segment, in fact), which restricts its application. Secondly, you would have to reasonably expect that the segment would arrive with no or correctable

errors. These conditions make it fairly unlikely to be of use in lossy environments with medium to large messages.

5.7 Flexible protocols

Flexible protocols are those that are not proscribed to behave in exactly the same way all the time. Instead, they attempt to have features that could be useful in a variety of situations and leave it open to be dynamically tailored to the situation at hand.

5.7.1 CTP

The Configurable Transport Protocol (see Wong [WONG01]) espouses the belief that a flexible transport protocol that can be tailored to specific situations is a smarter way of approaching protocol design than to have several independent protocols that take little or no account of each other. CTP conceptualises transport layer services as being separately implemented in micro-protocols. These micro-protocols would be responsible for providing services such as reliability, security, ordering and ensuring the performance and timeliness of individual connections. It is envisaged that these micro protocols would be selectable and would respond to independently defined network events (such as congestion), which can occur regardless of micro-protocol selection. The individual micro-protocols can then interact through shared data, including actual sent and received segments themselves.

This is an interesting idea, but from the paper, it didn't really seem to be a fully implemented set of protocols. Simply more of a philosophy of protocol design. This idea is more fully realised in XTP.

5.7.2 XTP

The eXpress Transport Protocol, specified in XTP [XTPF98], is one of the few ideas for a transport protocol that actually moves away from TCP. XTP is meant to be a flexible, connection based protocol that can be configured for use in many different cases, rather than the "one size fits all" approach attempted by TCP, which can be seen to have numerous problems by the sheer volume of modifications suggested to TCP.

XTP largely separates data transfer from control information. Mechanisms for flow control, rate control, error control and acknowledgement services are explicitly passed in control packets when the situation calls for them. Each of these controls can be used independently of one another, in any combination. This allows XTP to be tailored to the particular type of transport service required (e.g. unreliable, rate controlled traffic or reliable traffic without flow control etc…) without having to define a whole new protocol.

Error control is achieved through giving a sequence number (64-bits) that can (on request) be acknowledged by the receiver. XTP contains provisions for both selective

acknowledgement (informing the sender of which ranges of bytes the receiver has got) and negative acknowledgement (informing the sender of which bytes the receiver has not got).

Flow control (how much data you can send) is handled by a sliding window protocol whereby each side advertises (on request) to the other how much data it is willing to accept. Rate control (how quickly you can send data) is negotiated between end hosts using control packets that include desired and acceptable data rates and burst sizes.

XTP is a little more flexible than TCP in that it can support many different network types and can be extended to include more. It can even operate directly over data link layers such as Ethernet. XTP also contains explicit support for prioritising traffic and can inherently support one transmitter to many receiver multicast traffic. It even gives limited support to higher layers to allow them to mark "special" data for specific use by the application (e.g. it might be used to delimit application messages in what would otherwise be just a byte stream).

The flexibility of this protocol will allow it to be useful for many different situations. It tends to have rather large headers and could require extensive state information, but these would quite possibly not matter if there were sufficient performance gains. Unfortunately, XTP still requires only end-hosts to control their data rates, which might limit its effectiveness. On the other hand it certainly makes XTP more deployable. This is definitely worth further consideration.

5.8 Long delay networks.

The following protocols have been designed for networks paths with very high latencies.

5.8.1 SCPS-TP (TCP)

The Space Communications Protocol Specification (SCPS) does not define a single protocol but rather a collection of protocols designed to work together, intended for links that involve satellites. These include File (transfer), Security, Network and Transport Protocols. Only the transport protocols (SCPS-TP) will be discussed here. SCPS-TP has two parts: a modified version of UDP, which is nothing more than an optimisation of UDP to work with the SCPS network layer and a modified version of TCP, which is given an overview here. The details can be found in [CCSDS99].

The SCPS-TP TCP is designed to be completely compatible with existing TCP implementations. As such, most of the capabilities in this protocol take the form of TCP options and in general, it "looks" like TCP with an extra helping of capabilities. This TCP has facility for sending compressed headers as well as allowing the use of Selective Negative Acknowledgements (SNACK's) for a more efficient re-transmission strategy. It also provides a mechanism for indicating message boundaries within the byte stream, which removes the responsibility from the user to insert their own logic, as such it offers a message-oriented service. The specification suggests the use of a TCP Vegas style

congestion control algorithm, although the standard Jacobson algorithm or indeed no congestion control at all is also allowed. Another difference in this version of TCP is the best-effort transport service. This service allows the full reliability of TCP to be scaled down so that there is a limit to the number of times that any particular segment will be re-transmitted. This is intended for applications where data loses its value as it gets older. There is another optional capability that attempts to anticipate loss through the communication channel. Basically, each segment that needs to be transmitted can be transmitted several times without waiting to see if the first transmission gets lost or corrupted. This feature does not interoperate with any form of congestion control.

As transport protocols SCPS-TP do not really stand on their own as they are designed to work with the other protocols at different levels defined by the SCPS. Some of the benefits of SCPS-TP would only come from operating in this environment. Although they purport to be able to interoperate in an environment that contains more traditional internet protocol implementations, it is not clear what benefits can be obtained from SCPS-TP in this situation. Indeed, the main benefits from SCPS-TP over TCP appear to come in high error, high bandwidth-delay environments (which incidentally, is what SCPS-TP is designed for) where the increase in performance stems from simply not using the standard TCP congestion control algorithms, which are known to perform badly in this situation. (See [MUHO98] for a comparison of TCP and SCPS-TP performance.) Overall, the improvements to TCP in this case are relatively minor.

5.8.2 TCP Peach

This variant of TCP attempts to solve the problems TCP has with satellite links – i.e. long latencies and high error rates relative to wired links. [AKYI01] notes that the slow start algorithm can take a long time to take full advantage of satellite capacity because of the extremely long round trip times typical of satellite communication. TCP Peach attempts to remedy this by modifying slow start and fast recovery. Within the first round trip time in either of these algorithms, in addition to the data packets TCP could otherwise send, Peach sends additional *dummy* packets, up to the amount of bandwidth Peach believes can be used. The dummy packets carry no data and are very low priority. This means that the dummy packets will be dropped before any other traffic, and so won't impact on other network users. By checking the number of dummy packets that are acknowledged, Peach can reasonably accurately judge the bandwidth that it can use, and jump to sending that amount directly, rather than waiting for the senders window to work its way back up to size. Of course, this scheme relies on the underlying network supporting some form of priority-based traffic.

This is an innovative way to address rate control in TCP, but it still allows fluctuating congestion windows and leads to complicated implementations. It could also potentially be very unfair if congestion causes many connections to send dummy packets at the same time.

5.9 Networks with Single Wireless Links

The following TCP variations and other methods were designed to allow more effective operation over wireless links. Typically these protocols have been designed with a particular situation in mind. That situation is the one where the network topology has only single wireless links to mobile hosts at the very edge of the network infrastructure (similar to mobile phone networks). As such, some of these protocols (not all) are really only appropriate for this specific situation.

5.9.1 TCP Westwood

This is another TCP that can best be described as Reno with one small modification. This TCP tries to move away from the somewhat arbitrary halving of the slow start threshold when segments are lost. Instead, TCP Westwood maintains an ongoing estimate of the data rate that any particular connection is extracting from the network. When congestion is inferred through either segment timeout or receiving duplicate acknowledgements, the slow start threshold is set to the estimated data rate times the estimated round trip time. As this takes less dramatic action in the face of segment losses, it is expected that this TCP will be more suitable for use over wireless links. Results from [GERL01] show TCP Westwood significantly outperforming TCP Reno in situations with a segment error probability ranging from 0.0001 to 0.01.

This is a refinement of TCP that, while shown to be an improvement, probably doesn't go far enough in addressing the problems of TCP to be useful.

5.9.2 TCP Jersey

TCP Jersey, described in [XUTI04], uses a similar technique to that of TCP Westwood to estimate the bandwidth available to individual connections to maintain efficient data throughput. In addition, it makes an attempt to distinguish between segment losses due to congestion, and those due to link errors, with a view to improving TCP performance in environments with mixed wired/wireless links. It does this by relying on ECN to determine the cause of lost packets. TCP Jersey will not reduce the sender's window if it detects lost segments through the receipt of duplicate acknowledgements when it has not also received congestion warnings. Instead, it will assume the segment was lost due to link errors and re-transmit without slowing down its data-rate.

As a refinement of TCP Westwood, TCP Jersey further complicates TCP implementations, but it would be interesting to see how it performs against other non-TCP based protocols.

5.9.3 I-TCP

Indirect TCP, described by [BAKR95], attempts to solve the problems that arise when an ordinary TCP is used over a network path that contains both wired and wireless links. It treats the two links as different problems and splits the connection into two parts, namely

the wired part, and the wireless part. An I-TCP connection can then use ordinary TCP on the wired part, and a specialised wireless protocol (not specified) for the wireless link. One of the key goals of this approach is to make sure that no modifications are necessary to the existing hosts on wired networks. To achieve this, I-TCP requires the threshold router (the last router on the wired network leading to the wireless links) to essentially fool the host on the wired network into believing it has a direct connection with the host on the wireless network, which is actually not true. It also seeks to maintain connections when the host on the wireless network is mobile, and so can be disconnected for short periods of time or even change its network location altogether. Therefore I-TCP requires a complex procedure for transferring the duties of the old threshold router to the new threshold router, without the wired host ever needing to know anything unusual is going on.

I-TCP is not a complete solution by itself and could potentially be very complicated to implement, especially since connections over the wireless network are unlikely to behave in a similar way to wired connections. It could however, remove some of the problems of wireless links. You would expect it to be useful only if there are very few wireless links in your overall network. The benefits and problems with I-TCP would be the same with any solution that uses an intermediary host to manage the connection.

5.9.4 TCP Freeze

This variant (see [GOMO00]) is another attempt to address wired hosts communicating with – possibly mobile – hosts over a network with a wireless link. It tries to stay away from the common approach of modifying sender's communications stacks or requiring changes to intermediate routers. Instead, all it requires is one small change to the mobile host. The host is called mobile here and not simply "the host on the wireless network" because TCP-Freeze attempts to solve the problem of wireless hosts moving – namely signal fading – and not the problem of generally lossy links. There exists a mechanism in standard TCP whereby a receiver can instruct the sender to not send any more data. It does this by sending an acknowledgement advertising a window size of zero. TCP-freeze attempts to avoid losing packets altogether by sending this advertisement whenever it predicts that wireless conditions will soon become unfavourable. For example, a mobile host could monitor the signal strength and when signal strengths fall, it could inform the sender to stall data transfer until the signals start to become strong again, then send the proper window size advertisement to the sender. This will avoid the congestion control algorithms in the sender altogether. This does, of course, require some cross-layer communication, however.

This is an interesting idea that of course does not solve all of the problems of the MTCE, but certainly should be considered for integration into any other solution. It seems to be the only attempt to solve link errors at a level below the broad classification of lost data as an "error".

5.9.5 M-TCP

M-TCP is a type of I-TCP that uses the same trick as TCP-Freeze. Instead of getting the mobile host to send the zero window advertisement, however, M-TCP gets the threshold router to fool the wired host into believing a zero window advertisement has been sent when the threshold router detects that the mobile host has *actually* disconnected from the network (hopefully temporarily).

5.9.6 WTCP

Wireless Transmission Control Protocol is a significant variation of TCP that is designed for wireless wide area networks. These are essentially still topologies that have only one wireless link, but the wireless link tends to be responsible for a majority of the latency experienced by applications connecting across that link (see [SINH01]). WTCP is an end-to-end only mechanism. It does not require support from intervening routers.

Not only do the wireless wide area links experience low data capacity and high latency, but also highly varying data capacity and latency. This means that TCP's retransmission timeouts become almost impossible to use accurately, particularly since retransmission timeout estimates are further degraded by the bursty traffic flow that is typical in a wireless environment. To combat both unnecessary re-transmissions and fallacious conclusions of congestion that would result from timeouts, WTCP dispenses with that timer altogether. For reliability, WTCP uses periodic selective acknowledgements instead.

The WTCP version of congestion control is quite different to standard TCP though. It introduces rate control to the sender. Each sender transmits data at an advertised rate that may be checked and controlled by the receiver when it sends back acknowledgements. Having the receiver control the sender's rate in this way avoids the assumption that the return path (of acknowledgements) shares the same characteristics of the forward path (of data). The receiver checks the traffic rate by measuring the ratio of the inter-packet arrival time at the receiver to inter-packet sending time at the sender. When congestion becomes imminent, this period will become longer, and the traffic rate can be altered accordingly through the receiver's acknowledgements. WTCP can also distinguish between congestion related packet losses and random losses by checking the inter-packet delay of packets arriving close to the lost ones (while taking into account the missed packets making the time between packets longer).

WTCP appears to be a well thought out variation of TCP for its intended environment. Its applicability to the MTCE would need to be validated, particularly to check that the congestion control mechanisms would be effective in the MTCE, as they have the potential to outclass TCP's mechanisms, bur were explicitly designed for the WWAN environment.

5.9.7 Snoop

The Snoop idea, presented in [BALA95], attempts to solve the problem of introducing a lossy wireless link to a TCP connection by completely hiding the lossy nature of the link. It requires a modification to the router on the threshold between wired and wireless networks. This router will observe TCP traffic passing through it. If it becomes aware that segments have been lost between itself and the wireless host, it will quickly retransmit the lost segment. It will also stop duplicate acknowledgements coming back from the wireless host that would otherwise be incorrectly interpreted as congestion by the sender.

Of course, this scheme only works for data going in one direction and hence would need to be implemented on the wireless host as well.

This would only be useful as an optimisation to allow single hop wireless. It is doubtful that this would be appropriate for the MTCE.

5.9.8 The Reliable Data-Link Layer

It can be seen (from above) that a considerable number of schemes exist to improve TCP performance on network paths that include at least one wireless link. While not a transport protocol itself, it is envisioned that improving the reliability of the data-link layer (where, after all, the problems with wireless links exist) would allow large improvements in the performance of some transport protocols over those links. Several methods, such as employing an acknowledgement service at this layer, or using forward error correction, could be used to improve reliability. [BALA97] gives a good overview of this and other techniques used to solve this single wireless link situation, and compares relative performance of some of these schemes.

Link layer remedies would appear to be excellent solutions to the loss problem, though some of them will introduce delay variations. To use these schemes effectively, you would need a transport protocol that did not assume consistent timing for packet delivery though, which rules out standard TCP. This technique could probably be integrated with a TCP variation that exclusively used SACK for reliability.

Link layer solutions may help solve or alleviate the problems, but cannot replace end-to-end transport protocols.

5.10 Elephants

The following protocols were designed for very high speed networks, with large capacities, on which traditional TCP implementations also suffer (These networks are sometimes called Long Fat Networks, LFN's or Elephants). These sorts of networks are quite different to those found in the MTCE, therefore it is less likely to find ideas in these protocols that are useful to the MTCE.

5.10.1 Explicit Control Protocol

The eXplicit Control Protocol (XCP) described by [KATA02], cannot be wholly described as a variant of TCP, because it treats congestion as a network layer problem as well as a transport layer problem. It was originally designed to overcome problems TCP has with large data rates combined with long round trip times. To operate correctly, it requires routers on the flows of XCP traffic to be XCP aware. XCP differs from TCP mainly in its congestion control mechanisms. Instead of using the variety of solutions proposed over time, XCP uses its own system, based on ECN and control theory. A lot of the problems with TCP in differing network conditions come out of TCP essentially having to guess network conditions based on the behaviour of TCP packets, which was always less than ideal. With XCP, each router is responsible for controlling the flow through it. XCP achieves this by adding a parameter on each packet (in its header) representing the amount by which the sender's window should be decreased (or increased). Each router modifies this parameter based on its own congestion levels and perceived fairness to other flows of data. This value gets echoed back to the sender through an acknowledgement. This allows XCP to utilise available bandwidth more effectively and more fairly than TCP can achieve. XCP claims to be able to control traffic to such an extent that it will very rarely, if ever, drop packets due to congestion. This means it might also be far more suitable for wireless links than a standard TCP as losses can then be assumed to be from link errors and not congestion. XCP connections can also co-exist efficiently and fairly with TCP connections.

XCP addresses congestion control and fairness in a way that should match true rate control for effectiveness. However, it would require further modifications to the basis of its algorithm (TCP) for it to be useful in lossy and high latency environments. Its mechanisms should certainly be considered though.

5.10.2 HighSpeed TCP

In order to achieve steady-state transmission of very high data rates, TCP (in practise, not in definition of protocol) requires sufficiently low loss rates for TCP's standard congestion avoidance mechanisms to work efficiently (see [RFC3649]). In short, for a given loss rate (Note the losses don't have to be caused by congestion here), there is a limit on the effective throughput of TCP. HighSpeed TCP attempts to address this issue by adjusting the congestion control mechanisms to be more suited to high data rate environments. Specifically, it strengthens the relation between increase and decrease of the sender's window to the current window size. TCP Reno used an increase in windows size of one segment per round trip time (in the congestion avoidance stage), and a decrease in window size of 0.5 of the current window when congestion is inferred. Highspeed TCP uses an increase of **a(w)/w** per round trip time, where **w** is the window size, and **a(w)** is a function that increases with window size. On congestion, the window size is reduced to **(1-b(w))w** where **w** is the old window size and **b(w)** is a function that decreases with w and has a range from 0.0 to 1.0. See [RFC3649] for a more complete analysis of the choice of values.

This variation of TCP doesn't address any of the problems with the MTCE and so would be unsuitable.

5.10.3 Scalable TCP

Scalable TCP (see [KELL03]) is a TCP modification designed for wide area networks that have long round trip times and high data rates. It attempts to improve the rate at which TCP recovers from congestion detection by increasing the rate at which the sender's congestion window grows during congestion avoidance, and decreasing the rate at which the senders congestion window shrinks when congestion is inferred. Instead of increasing by one segment every round trip time (or approximately 1/window size for each acknowledgment received) Scalable TCP increases the windows size by 0.01 for each acknowledgment received. Also, instead of decreasing by 0.5 * window size on congestion, Scalable TCP decreases by 0.125 * window size. These modifications basically make TCP more aggressive, which is seen as desirable for high speed wide area networks used for bulk data transfer.

This could only ever be appropriate for its intended environment, which is unlike the MTCE and doesn't address any of the other problems with TCP.

5.11 Multicast Considerations

Sometimes transport protocols have built into them explicit support for multicasting messages. An example of this would be XTP (see section 5.7.2). While in some situations this might be a desirable feature, it can also generally be added on top of an existing unicast only transport protocol. [ANUP02] provides a method for doing this in ad hoc networks. A very brief description is given here purely for illustrative purposes to show that multicast capabilities can be added on to existing working transport protocols.

This protocol is a system for constructing and maintaining multicast trees inside ad hoc networks. Group join, leave and error messages are propagated through the network with a broadcast mechanism, to enable nodes to form knowledge of their upstream and downstream neighbours in the multicast tree. If this information is already available, then each node can make more intelligent decisions about where to send individual group messages. This is not a true transport protocol, it is more of a multicast add-on which would operate over the top of a unicast transport protocol.

6. Conclusion

The design of existing transport protocols has been driven by the most widely used forms of networks, and hence are usually unsuitable for meeting the needs of the MTCE, which have significantly different characteristics. There are a wealth of optimisations and add-

ons for existing protocols as well as entirely new ones that have been designed to solve various problems with commonly used transport protocols (most notably TCP). Some of these show the potential to be useful in the MTCE and should be investigated to ascertain if they can meet the needs of this environment. These include ATP, RBUDP, XTP and WTCP, and certainly there are methods in other protocols that bear consideration as well such as the rate control in XCP, various reliable link layer methods and the one way characteristics of P_MUL. It is also possible that very thin workarounds like the method put forward in ATCP could be all that is required.

7. References

[ACP142] "P_MUL – A Protocol for Reliable Multicast Messaging in Bandwidth Constrained and Delayed Acknowledgement (EMCON) Environments", ACP142, December 2001.

[AKYI01] I.F. Akyildiz, G. Morabito and S. Palazzo. "TCP-Peach: A new congestion control scheme for satellite IP networks", IEEE/ACM Trans. Networking, Volume 9, pp 307-321, June 2001.

[ANAS03] G. Anastasi and A Passarella, "Towards a Novel Transport Protocol for Ad Hoc Networks", 2003.

[ANUP02] K.R. Anupama and S. Balasubramanian, "A Multicast Protocol for Mobile Adhoc Networks", IEEE International Conference on Personal Wireless Communications, pp 187-191, December 2002.

[BAKR95] A Bakre and BR Badrinath, "I-TCP: Indirect TCP for mobile hosts", Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS), pp. 136-143, May 1995.

[BALA95] H. Balakrishnan, S Sheshan and RH Katz, "Improving reliable transport and handoff performance in cellular wireless networks", ACM Wireless Networks Volume 1, December 1995.

[BALA97] H. Balakrishnan, VN Padamanabhan, S Sheshan and RH Katz, "A comparison of Mechanism for Improving TCP Performance over Wireless Links", ACM/IEEE Transaction on Networking, Volume 5, Number 6, pp 756-769, December 1997.

[BRAK95] L. Brakmo, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", IEEE Journal on Selected Areas in Communications, Volume 13, Number 8, October 1995.

[CCSDS99] "Space Communications Protocol Specification (SCPS) Transport Protocol (SCPS-TP) – Blue Book", CCSDS Secretariat – NASA, Washigton, May 1999.

[FALL96] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno and SACK-TCP", Computer Communications Review, July 1996 - ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z.

[FULI03] Cheng Peng Fu, S.C. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks", IEEE Journal on Selected Areas in Communication, Volume 21, Issue 2, February 2003.

[GERL99] Mario Gerla, Ken Tang and Rajive Bagrodia, "TCP Performance in Wireless Multihop Networks", Proceedings of IEEE WMCSA '99, February 1999.

[GERL01] M. Gerla, M.Y. Sanadidi, Ren Wang, A. Zanella, C. Casetti, S. Mascolo, "TCP Westwood: congestion window control using bandwidth estimation", Global Telecommunications Conference 2001, GLOBECOM '01, IEEE Volume 3, 25-29 November 2001.

[GOMO00] T. Go, J. Moronski, D. S. Phatak, and V. Gupta, "Freeze-TCP: A true end-to-end enhancement mechanism for mobile environments", INFOCOM, Israel, 2000.

[HELE02] Eric He, Jason Leigh, Oliver Yu and Thomas A. DeFanti, "Reliable Blast UDP: Predictable High Performance Bulk Data Transfer:, Proceedings of IEEE Cluster Computing, Chicago Illinois, September 2002.

[JACO88] Van Jacobson, "Congestion Avoidance and Control", Proceedings of SIGCOMM '88, Stanford CA, Aug 1988.

[JACO90] Van Jacobson, "Modified TCP Congestion Avoidance Algorithm", email to the end2end list, April 1990.

[KARN87] P. Karn and C. Partridge, "Estimating Round-Trip Times in Reliable Transport Protocols", Proceedings of SIGCOMM '87, Stowe VT, August 1987.

[KART03] Karthikeyan Sundaresan, Vaidyanathan Anantharaman, Hung-Yun Hsieh and Raghupathy Sivakumar, "ATP: A Reliable Transport Protocol for Ad hoc Networks", MobiHoc '03, 2003.

[KATA02] D. Katabi, M. Handley and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks", ACM SIGCOMM, 2002.

[KELL03] Tom Kelly, "Scalable TCP: Improving Performance in High Speed Wide Area Networks", First International Workshop on Protocols for Fast Long-Distance Networks, Geneva, February 2003.

[KENN97] Alan Ulrich Kennigton, "The FITFEEL Transmission Protocol", ICT '97, Melbourne, pp 1391-1397, April 1997.

[KHUU04] Binh Khuu and Perry Blackmore, "Transmission Control Protocol (TCP) Performance in a Satellite Multiple User Access System", DSTO-TN-0579, Edinburgh South Australia, July 2004.

[LAKS97] T.V. Lakshman and U. Madhow, "The Performance of TCP/IP for networks with high bandwidth-delay products and random loss", IEEE/ACM Trans.Networking, Volume 5, pp 336-350, June 1997.

[LEFE00] F. Lefevre and G vivier, "Understanding TCP's behaviour over wireless links", Proceedings of Communications Vehicular Technology, SCVT-2000, pp. 123-130, 2000.

[LIUS01] J. Liu and S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks", IEEE Journal on Selected Areas in Communications, Volume 19, Issue 7, pp 1300-1315, July 2001.

[MICR00]
http://www.microsoft.com/windows2000/techinfo/howitworks/communications/networkbasics/tcpip_implement.asp

[MICR03]
http://www.microsoft.com/technet/prodtechnol/windowsserver2003/technologies/networking/tcpip03.mspx

[MUHO98] John Muhonen and Robert C. Durst, "Space Communications Protocol Standards (SCPS) FY97 DOD Test Report", MITRE Corporation, 1998.

[RAMA00] R. Ramani and A. Karandikar, "Explicit Congestion Notification in TCP over Wireless Networks", IEEE Conference on Personal Wireless Communications, pp 495-499, 17-20 December 2000.

[RFC793] Marina del Rey, "RFC 793: Transmission Control Protocol", Internet RFC's, 1981.

[RFC 813] D. Clark, "RFC 813: Window and Acknowledgment Strategy in TCP", Internet RFC's, July 1982.

[RFC896] John Nagle, "RFC 896: Congestion control in IP/TCP internetworks", Internet RFC's, January 1984.

[RFC1072] Van Jacobson, R Braden, "RFC 1072: TCP Extensions for Long Delay Paths", Internet RFC's, October 1988.

[RFC1106] R. Fox, "RFC 1106: TCP Big Window and NAK Options", Internet RFC's, June 1989.

[RFC1644] R. Braden, "RFC1644: T/TCP – TCP Extensions for Transactions", Internet RFC's, July 1994.

[RFC2581] M. Allman, V. Paxson, W. Stevens, "RFC 2581: TCP Congestion Control", Internet RFC's, April 1999.

[RFC2582] S. Floyd and T Henderson, "RFC 2582: The NewReno Modification to TCP's Fast Recovery Algorithm", Internet RFC's, April 1999.

[RFC2914] S. Floyd, "RFC 2914: Congestion Control Principles", Internet RFC's September 2000.

[RFC2960] R. Stewart et al., "RFC 2960: Stream Control Transmission Protocol", Internet RFC's, 2000.

[RFC3286] L. Ong, J. Yoakum, "RFC 3286: An Introduction to the Stream Control Transmission Protocol (STCP) ", Internet RFC's, May 2002.

[RFC3649] S. Floyd, "RFC 3649: HighSpeed TCP for Large Congestion Windows", Internet RFC's, December 2003.

[SARO02] P. Sarolahti and A. Kuznetsov, "Congestion Control in Linux TCP", Proceedings of Usenix 2002 / Freenix Track, pp 49-62, Monterey USA, June 2002, http://www.cs.helsinki.fi/research/iwtcp/papers/linuxtcp.pdf.

[SINH01] Prasun Sinha, Thyagarajan Nandagopal, Narayanan Venkitaraman, Raghupathy Sivakumar and Vaduvur Bharghaven "WTCP: a reliable transport protocol for wireless wide-area networks", Wireless Networks, Volume 8, Issue 2/3, March 2002.

[TANE03] Andrew S Tanenbaum, "Computer Networks", Prentice Hall, New Jersey, 2003.

[TOZH04] J. Tovirac and W. Zhang, Enhancement schemes to P_Mul multicast protocol, 3rd workshop on Internet, telecommunications and signal processing, WITSP'2004, 22-24 Dec. 2004, Adelaide

[WONG01] G.T. Wong, M.A., Hiltunen and R.D. Schlichting, "A Configurable and Extensible Transport Protocol", Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, Volume 1, pp 319-328, April 2001.

[XTPF98] XTP Forum, "Xpress Transport Protocol Specification 4.0b", July 1998, http://www.mentat.com/xtp/XTP40b.pdf.

[XUTI04] Kai Xu, Ye Tian and Nirwan Ansari, "TCP-Jersey for Wireless IP Communications", IEEE Journal on Selected Areas in Communications, Volume 22, Issue 4 pp 747-756, May 2004.

[ZHEN02] Zhenghua Fu, Benjamin Greenstein, Xiaoqiao Meng, and Songwu Lu, "Design and Implementation of a TCP-Friendly Transport protocol for Ad Hoc Wireless Networks", Proceedings of the 10th IEEE International Conference on Network Protocols, 2002.

Appendix A: Socket Types

The traditional API for programs communicating over networks is the Berkeley sockets interface. This generally involves a program opening a "socket" while supplying three parameters to totally specify the protocols that will be used. The parameters are *domain, type* and *protocol*. The *domain* is not greatly relevant to the transport protocol as it generally indicates what type of network the socket will be communicating over. The *type* however, is an important classification of the type of service that the transport protocol will provide. The *protocol* is just a number to differentiate between protocols if more than one protocol can offer a similar type of service. The following is a description of the various socket *type*s and a classification of the discussed transport protocols into their appropriate *type*s.

A.1. Stream

A Stream type protocol provides reliable communication of data in which order of bytes is preserved in a stream-oriented fashion. That is, a user will only see a stream of incoming and outgoing bytes, and not be aware of the segments that the actual data is transmitted in. Stream protocols provide two-way communication within the context of a connection.

A.2. Dgram

A protocol with type Dgram does not guarantee the delivery, uniqueness or order of transmitted segments. That is, it provides an unreliable, best-effort service only. Dgram protocols are message-oriented. That is, data is passed to the transport service in discrete chunks of data that are then delivered to their destination in exactly the same chunks. Dgram protocols are connectionless.

A.3. Rdm

Protocols of type Rdm are similar to Dgram protocols. They provide a connectionless, message-oriented service. Unlike Dgram, Rdm protocols provide reliability as well.

A.4. Seq_packet

Seq_packet protocols are connection based protocols that are also message-oriented. They provide a reliable, order preserved sequence of segments between two hosts.

A.5. Discussed Transport protocols

The following is a classification of the transport protocols into the Berkeley socket types. Note that the *type*s are only classifications, not specifications that protocols must abide by. As will be seen, not all protocols can be fitted to this model, and not all of them are restricted to only one *type*. Nevertheless an attempt has been made to fit most of them into the *type* that is likely to suit that particular protocol best.

Stream	Dgram	Rdm	Seq_packet	Unclassifiable
TCP TCP NewReno TCP Vegas ADTCP ATP TPA ATCP T/TCP TCP Westwood TCP Jersey I-TCP TCP Freeze M-TCP WTCP TCP-Peach XCP HighSpeed TCP Scalable TCP SCPS-TP		FITFEEL P_MUL	SCTP	RBUDP CTP XTP Snoop R. data link

RBUDP cannot be classified because it uses a combination of reliable connection oriented traffic and unreliable connectionless traffic. So it is really both Stream and Dgram. CTP and XTP are by their very nature configurable, and so do not restrict themselves to any one of the types indicated here. Snoop and the reliable data link are approaches that do not modify the transport protocol (they modify lower layers instead), and hence have no applicability in this sort of classification.

DISTRIBUTION LIST

A Review of Transport Protocols as Candidates for Use in a Tactical Environment

M. Carter

AUSTRALIA

DEFENCE ORGANISATION	**No. of copies**
Task Sponsor	
Director General Integrated Capability Development	1 Printed
S&T Program	
Chief Defence Scientist	1
Deputy Chief Defence Scientist Policy	1
AS Science Corporate Management	1
Director General Science Policy Development	1
Counsellor Defence Science, London	Doc Data Sheet
Counsellor Defence Science, Washington	Doc Data Sheet
Scientific Adviser to MRDC, Thailand	Doc Data Sheet
Scientific Adviser Joint	1
Navy Scientific Adviser	Doc Data Sht & Dist List
Scientific Adviser – Army	1
Air Force Scientific Adviser	Doc Data Sht & Dist List
Scientific Adviser to the DMO	Doc Data Sht & Dist List
Information Sciences Laboratory	
Chief of Information Networks Division	Doc Data Sht & Dist List
Research Leader Military Communications	1
Head Mobile Networks Group	1
Head Wireless Systems Group	1
Head Network Management Group	1
Dr P.A. Blackmore	1
M. Hue	1
W.D. Blair	1
M. Carter	1 Printed
DSTO Library and Archives	
Library Edinburgh	1 printed
Defence Archives	1 printed

Library Canberra	1 printed
Capability Development Group	
Director General Maritime Development	Doc Data Sheet
Director General Land Development	1
Director General Capability and Plans	Doc Data Sheet
Assistant Secretary Investment Analysis	Doc Data Sheet
Director Capability Plans and Programming	Doc Data Sheet
Deputy Director Information Networks (Ms. Tina Ormsby)	1
Deputy Director Mobile Communications	1
Deputy Director Long Range Communications	1
SO Mobile Communications – Land	1
SO Mobile Communications – Maritime	1
Chief Information Officer Group	
Director General Australian Defence Simulation Office	Doc Data Sheet
AS Information Strategy and Futures	Doc Data Sheet
Director General Information Services	Doc Data Sheet
Strategy Group	
Director General Military Strategy	Doc Data Sheet
Assistant Secretary Governance and Counter-Proliferation	Doc Data Sheet
Navy	
Maritime Operational Analysis Centre, Building 89/90 Garden Island Sydney NSW Deputy Director (Operations) Deputy Director (Analysis)	Doc Data Sht & Dist List
Director General Navy Capability, Performance and Plans, Navy Headquarters	Doc Data Sheet
Director General Navy Strategic Policy and Futures, Navy Headquarters	Doc Data Sheet
Air Force	
SO (Science) - Headquarters Air Combat Group, RAAF Base, Williamtown NSW 2314	Doc Data Sht & Exec Summary
Army	
Director General Future Land Warfare	Doc Data Sht & Exec Summary
Director Network Centric Warfare (NCW) – Army	1
SO1 CISEW (Force Development Group), Land Warfare Development Centre, Puckapunyal	1
ABCA National Standardisation Officer Land Warfare Development Sector, Puckapunyal	e-mailed Doc Data Sheet
SO (Science) - Land Headquarters (LHQ), Victoria Barracks NSW	Doc Data & Exec Summary
SO (Science), Deployable Joint Force Headquarters (DJFHQ) (L), Enoggera QLD	Doc Data Sheet

Joint Operations Command

Director General Joint Operations	Doc Data Sheet
Chief of Staff Headquarters Joint Operations Command	Doc Data Sheet
Commandant ADF Warfare Centre	Doc Data Sheet
Director General Strategic Logistics	Doc Data Sheet
COS Australian Defence College	Doc Data Sheet

Intelligence and Security Group

AS Concepts, Capability and Resources	1
DGSTA , Defence Intelligence Organisation	1 Printed
Manager, Information Centre, Defence Intelligence Organisation	1
Director Advanced Capabilities	Doc Data Sheet

Defence School of Signals

Commandant, Defence Force School of Signals, Simpson Barracks, Macleod, Vic., 3085	Doc Data Sht & Exec Summary
Commandant, Defence Force School of Signals, Simpson Barracks, Macleod, Vic., 3085	Doc Data Sht & Exec Summary

Defence Materiel Organisation

Deputy CEO	Doc Data Sheet
Head Aerospace Systems Division	Doc Data Sheet
Head Maritime Systems Division	Doc Data Sheet
Program Manager Air Warfare Destroyer	Doc Data Sheet
CDR Joint Logistics Command	
Guided Weapon & Explosive Ordnance Branch (GWEO)	Doc Data Sheet
PD JP 2072	1
PD SEA 1442	1
PD JP 2043	1
PD JP 2008	1
Land Engineering Agency	
Library	1
Mr G. Lampard	1

OTHER ORGANISATIONS

National Library of Australia	1
NASA (Canberra)	1

UNIVERSITIES AND COLLEGES

Australian Defence Force Academy

Library	1
Head of Aerospace and Mechanical Engineering	1
Hargrave Library, Monash University	Doc Data Sheet

OUTSIDE AUSTRALIA

INTERNATIONAL DEFENCE INFORMATION CENTRES	
US Defense Technical Information Center	1
UK Dstl Knowledge Services	1
Canada Defence Research Directorate R&D Knowledge & Information Management (DRDKIM)	1
NZ Defence Information Centre	1
ABSTRACTING AND INFORMATION ORGANISATIONS	
Library, Chemical Abstracts Reference Service	1
Engineering Societies Library, US	1
Materials Information, Cambridge Scientific Abstracts, US	1
Documents Librarian, The Center for Research Libraries, US	1
SPARES	5 Printed

Total number of copies: Printed: 11 PDF:41

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

A Review of Transport Protocols as Candidates for Use in a Tactical Environment

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	U
Title	U
Abstract	U

4. AUTHOR(S)

Michael Carter

5. CORPORATE AUTHOR

Defence Science and Technology Organisation
PO Box 1500
Edinburgh South Australia 5111 Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-TR-1808	AR-013-549	Technical Report	December 2005

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
	JTW 02-098	DGICD	40	50

13. URL on the World Wide Web

http://www.dsto.defence.gov.au/corporate/reports/ DSTO-TR-1808.pdf

14. RELEASE AUTHORITY

Chief, Information Networks Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS Yes

18. DSTO Thesaurus

Transport protocols, Tactical communications, Communications networks

19. ABSTRACT

Transport protocols are an essential part of any modern data network, including military tactical networks. Most networks today, including the Internet, operate mainly using the transport protocol TCP. However, this protocol makes a number of assumptions that make it unsuitable for use in a military environment, which is typically characterised by data links that are impoverished in some way. This paper examines TCP, its deficiencies and many other transport protocols that are evaluated for suitability to a military tactical environment. Many of the essential services of TCP as well as other desirable services can be well met by some of these other protocols in existence today in a way that is more suitable for a military tactical environment.